COMPUTER SYSTEM WITH GRAPHICAL USER INTERFACE INCLUDING SPRING-LOADED ENCLOSURES

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to computer systems with graphical user interfaces, such as window based systems; and more particularly to techniques for finding, moving, and copying objects in such systems.

Description of the Related Art

Graphical user interfaces on computer systems are gaining widespread use. Typical systems include the Macintosh Finder™ environment in Macintosh computers provided by Apple Computer, Inc., of Cupertino, California, the Windows environment provided by Microsoft Corporation of Redmond, Washington, and the New Wave™ environment provided by Hewlett-Packard of Palo Alto, California. In such systems, a workspace on the display system is set up with a desktop metaphor. Within the desktop, there are icons displayed which correspond to objects stored in memory. Many icons represent enclosures that enclose other objects. Opening the enclosure results in display of a window that encloses related icons.

These systems provide the ability to move an object from an enclosure represented by an icon within one window to an enclosure represented by another window or icon, or to copy an object represented by an icon within one window into an enclosure represented by another window or icon. These operations involve first setting up a destination window. The problem of setting up the destination window may be quite complicated, when the system involves a complex hierarchy of enclosures.

Further, the process of setting up the destination window may clutter the desktop with a number of windows, obscuring the windows actually in use.

After setting up the destination window, the cursor is moved to an icon representing the object to be placed in the destination window by the move or copy operation. A drag operation is then executed to move the icon into the destination window. The drag operation typically involves placing the cursor over the icon subject of the drag operation, depressing the mouse button, and while the mouse button remains depressed, moving the cursor into the destination window. The mouse button is released when the cursor is over the destination window. The operating system responds to this manipulation of the graphical interface by either moving the object into the destination window if both the object and the destination are stored on the same disk, or copying the object into the destination window if the destination is on a different disk than the object.

Finally, after the drag operation, the user must clean up the desktop by closing the unneeded windows that were opened during the process of setting up the destination window.

As can be appreciated, this process is quite cumbersome when the destination window is deep in a hierarchy. Also, the problem of setting up the destination window makes initiation of a drag operation problematic. It would be desirable to be able to browse through the storage system hierarchy after the drag operation has begun.

SUMMARY OF THE INVENTION

The present invention provides a new behavior in a graphical user interface which allows the user to open

and close enclosure windows, while dragging an object. When the user pauses, gestures, or rolls over a hot spot on top of an identifier representing a closed enclosure, a temporary window for the closed enclosure is "sprung open" to allow the user to browse inside the enclosure and possibly open another enclosure contained within the sprung open window. This process can be carried on throughout a deep hierarchy of windows as the user browses for a destination for the drag operation.

The user may close sprung open windows by simply moving the cursor out of the sprung open window, or by making some other indication with the pointing device, as suits the needs of a particular design.

Further, if an enclosure to be sprung open was previously open on the desktop, the previously opened window may zoom over to the current cursor position, and then return to its original position when the user moves the cursor out of the window.

Accordingly, the present invention can be characterized as an apparatus for finding or browsing among enclosures in a computer system which includes a display, a memory, and a pointing device with which the user drags a pointer (cursor) on the display. The apparatus includes a storage system that stores a plurality of objects which include enclosures in the memory. The enclosures comprise objects which may enclose other objects, and when opened, are represented by a window on the desktop. The system includes window management software that draws windows on the display corresponding to opened enclosures. Also, a mouse with a mouse button, or other input device is included which supplies a signal to indicate a drag operation. The spring loaded enclosures are managed with software responsive to a drag during a drag operation of the

pointer over an identifier (textual or graphical) corresponding to a particular enclosure for opening a temporary window for the particular enclosure to display icons within the temporary window that correspond to the objects enclosed by the particular enclosure. Further, the spring-loaded enclosure management software is responsive to a drag during the drag operation of the pointer outside the temporary window for closing the temporary window.

By releasing the mouse button or otherwise indicating an end of the drag operation, the user signals software for placing the particular object subject of the drag into the particular enclosure which has been sprung open during the drag.

The spring-loaded enclosure management software further determines whether the display includes an existing window opened for the particular enclosure during the drag operation to open a temporary window, and, if so, then removing the existing window from the display and drawing the temporary window on the display centered around the cursor, or otherwise associated with the position of the identifier corresponding to the particular enclosure. Also, in one aspect, the invention provides for graphically indicating on the display a zoom of the existing window over to the location of the temporary window. When the temporary window is closed, the existing window may be re-drawn on the display at its original position.

The decision to open a temporary window during a drag operation may be conditioned on actions by the user of the pointing device, such as pausing over the identifier for the particular enclosure, or making some other gesture. For instance, the identifier for enclosures that may be opened may include a hot region or

temporary window area. Moving the cursor over the temporary window area of the identifier will cause the enclosure to be sprung open. Alternatively, moving the cursor over an identifier of an enclosure may cause display of a split selector graphic. Moving the cursor to a particular side of the split selector graphic will cause the enclosure to spring open; while moving to the other side of the split selector graphic will cause the split selector graphic to be removed from the screen.

Thus, using the spring-loaded enclosure mechanism of the present invention, the user of a graphical user interface is free to browse through enclosures while dragging, rather than being forced to set up the source and destination before the drag begins. This greatly increases the ease of use of the graphical user interface.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a schematic diagram of a computer system implementing the spring-loaded enclosure management of the present invention.

Figs. 2A-2F illustrate a drag operation with spring-loaded enclosures according to the present invention.

Figs. 3A-3E illustrate alternative drag sequences using spring-loaded enclosures according to the present invention.

Fig. 4 is a block diagram of functional components of the user interface control system according to the present invention.

Fig. 5 is a schematic diagram of data structures used by the system of Fig. 4.

Figs. 6-13 provide flow charts for the operation of the system of Fig. 4.

Figs. 14A and 14B illustrate an alternative technique for indicating a wish to open a spring-loaded enclosure.

Figs. 15A and 15B illustrate another alternative technique for indicating a wish to open a spring-loaded enclosure.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

A detailed description of a preferred embodiment of the present invention is provided with respect to the figures. Fig. 1 provides a system overview. Figs. 2A-2D and 3A-3E illustrate the operation of the graphical user interface using spring-loaded enclosures. Figs. 4-13 illustrate an implementation of the control software for a system executing the present invention. Figs. 14A-14B and 15A-15B illustrate alternatives for opening spring-loaded enclosures.

I. System Overview (Figs. 1, 2A-2F, and 3A-3E)

Fig. 1 illustrates a computer system implementing the spring-loaded enclosure feature of the present invention. The computer system includes a host CPU 10 coupled to a system bus 11. The system includes a keyboard 12, a mouse 13 including a mouse button, or other pointing device, and a non-volatile memory 14, such as a hard disk, floppy disk, non-volatile integrated circuit memory systems, or the like. Similarly, instruction memory 15 and working memory 16 are coupled to the bus 11. The instruction memory 15 stores spring-loaded enclosure management software and window management software, among other software needed for operation of the system. The working memory 16 is used

to maintain a list of sprung open enclosures, and a window list or other tables needed by the software in the instruction memory 15.

Finally, the system includes a display controller 17 which includes video memory. The display controller 17 drives a display 18 such as a CRT video monitor, LCD flat panel display, or the like. The display system 18 has a screen, generally 19. On the screen 19, a workspace 20 is displayed. The workspace 20 is implemented with a desktop metaphor in the Macintosh type systems. Within the desktop 20, a plurality of identifiers may be displayed, such as the identifier 21 representing a hard disk drive, the identifier 22 representing a floppy disk, and other identifiers not shown which represent files, applications, control panels, or enclosures which enclose other objects. Also on the desktop 20, a plurality of windows, such as windows 23, 24, and 25 may be opened. The windows 23, 24, and 25 enclosed identifiers, such as identifiers 26 and 27 in window 23, identifiers 28, 29, and 30 in window 24, and identifier 31 in window 25.

In the figure, the identifiers are shown as graphical elements, or icons. Alternative identifiers may be textual elements, such as the name of the corresponding object. The behaviors described herein may be applied to both textual and graphical elements, as may occur in windows opened in a view by name mode or a view by icon mode in Macintosh computers.

In the illustration of Fig. 1, the desktop 20 is somewhat smaller than the screen 19. Alternative systems may extend the desktop metaphor to the entire area of the screen 19.

The spring-loaded enclosure management software and window management software in the instruction memory 15 of the system are used to open and close windows, and to

maintain records concerning the open and closed windows, and their positions within the desktop 20, the location of icons within the windows or on the desktop 20, and the like. During a drag operation, the spring-loaded enclosure management software in the instruction memory 15 is operable to create temporary windows so that the user may browse during the drag operation as described above.

Examples of the operation of the spring-loaded enclosures are provided with reference to Figs. 2A-2F and 3A-3E. In these diagrams, the same windows 23, 24, and 25 of Fig. 1 are used as a base for ease of understanding. The "star" identifier 30 in window 24 will be the particular object subject of the drag operation in the examples described.

Figs. 2A-2F illustrate a basic drag operations using spring-loaded enclosures. In Fig. 2A, the identifier 30 in window 24 is selected for a drag operation by moving the cursor 50 over the icon 30, depressing the mouse button and dragging the cursor along a path 51 while holding the mouse button down. The user pauses the cursor over identifier and causes a temporary window 52, shown in Fig. 2B, to open substantially centered over the cursor, and thus over identifier 27. Because the temporary window 52 is slightly too wide to open precisely centered over the cursor 27, it is redrawn within the desktop area as close as possible to the preferred location centered over the cursor. During the drag operation, the cursor carries an altered view 53 (e.g., an outline) of the star identifier 30 and is displayed within the temporary window 52. The drag operation continues along path 54 over identifier 55 within the temporary window 52. By pausing over the identifier 55, a temporary window 56, as shown in Fig.

2C, is opened, centered over the identifier 55. The altered view 53 of the identifier and the cursor now reside within the temporary window 56. The user then completes the drag operation to point 57 by releasing the mouse button. This results in placing the identifier 30 within the temporary window 56, as shown in Fig. 2D. Also, window 52 closes on mouse up, because it is not the destination. The identifier 30 is removed from window 24 if the temporary window 56 resides on the same disk as temporary window 24. Otherwise, the icon 30 is copied and will remain in both enclosures.

As shown in Fig. 2D, the temporary window 56 becomes a regular window on the display, as indicated by removal of the hatching across the top of the window 56. In implementation, temporary windows may be displayed in the same manner as other windows, or marked somehow as temporary. Also, at the end of the drag operation, all of the temporary windows, other than the destination window, are removed from the display. Thus, the temporary window 52 is no longer displayed within the desktop as shown in Fig. 2D.

Figs. 2E and 2F illustrate an alternative behavior. In this aspect, the temporary windows, such as window 52, do not automatically close after termination of the drag operation which leaves the star icon 30 in window 56. Rather, the temporary windows are closed in response to movement of the pointer after termination of the drag outside of the particular temporary window. Thus, as illustrated in Fig. 2E, if after termination of the drag, the user moves the cursor along path 58 outside temporary window 56, then the temporary window 56 will be removed from the screen. Temporary window 52 as illustrated in Fig. 2F remains on screen because the cursor remains within that temporary window. If the user then moves the

cursor along path 59 outside temporary window 52, then temporary window 52 will be removed, returning the screen to the configuration of Fig. 2A, except with the star icon 30 moved. The user, according to this aspect, may choose to maintain a temporary window open by clicking the mouse button while the cursor lies within the temporary window. Thus, if after completion of the drag operation in Fig. 2D, the user had clicked the mouse to select window 56 as a permanent window, and then moved the mouse out of window 56, such as along path 59, then the screen would be left in the configuration of Fig. 2D. Alternatively, the user could opt to maintain window 52 as a permanent window by clicking within the window when the cursor is positioned as shown in Fig. 2F.

Fig. 3A illustrates the beginning of an alternative sequence, using the spring-loaded enclosure system of the present invention. Fig. 3A correspond closely to Fig. 2C. However, in this instance, the user executes a drag operation along the path 60 which crosses the boundary of the temporary window 56 back into temporary window 52. When the boundary of temporary window 56 is crossed, the temporary window 56 is removed from the screen, as illustrated in Fig. 3B, and the altered view 53 of the identifier and the cursor reside within temporary window 52. If the user had moved the cursor along path 75, rather than path 60, temporary window 56 would close, but temporary window 52 would remain open. The user could close temporary window 52 by moving the cursor back into temporary window 52 and then back out as illustrated by path 76.

As shown in Fig. 3B, the user continues to browse by dragging along the path 61 to identifier 62. By pausing over the identifier 62, temporary window 63, as illustrated in Fig. 3C, is opened, centered over the

identifier 62. Also, the altered view 53 of the identifier and the cursor reside within temporary window 63.

Fig. 3C also illustrates what happens when the temporary window 63 corresponds to a window, e.g., window 25, which already existed on the desktop before the drag operation began. In this instance, the window 25 is removed from the screen. Also, a zoom operation indicated by the arrows 64 and 65 is graphically depicted on the screen to indicate to the user the movement of the window 25 to the temporary window 63. This zoom operation can take a variety of graphical characteristics difficult to show in the storyboards of Figs. 3A-3E. However, it will be understood by those skilled in the art how this operation is accomplished.

In Fig. 3C, the drag operation continues along path 66 to point 67, and the mouse button is released indicating the end of the drag operation. Fig. 3D illustrates one alternative outcome. In this case, the identifier is moved into the temporary window 63 and the temporary window is moved back to the original position of window 25. Window 25, including the identifier 30, remains on the screen. The identifier 30 has been removed from window 24 because windows 25 and 24 reside on the same disk. Alternatively, the user may have the option of keeping the temporary window 63 as the permanent window. This is illustrated in Fig. 3E. After the drag operation, the identifier 30 is left within the temporary window 63. The cursor 50 returns to its normal shape. The user has the option of closing the temporary window, which leaves the position of the real window unaffected. When the enclosure is opened again, the window will be located in its old location. If the user chooses not to close the temporary window but rather

moves or resizes the temporary window before closing it, when the window is subsequently re-opened it will be located at the new position and size.

In the illustrations of Figs. 2A-2D and 3A-3E, temporary windows are indicated by hatching a bar across the top of the window. In a preferred system, these windows may be rendered translucent, or other effects may be used as suits the needs of a particular implementation. As mentioned above, some means of indicating a temporary window, as opposed to a regular window, may be provided within the desktop.

II. Interface Management Logic Implementation (Figs. 4-13)

As mentioned with reference to Fig. 1, a computer system implementing the spring-loaded enclosure mechanism according to the present invention includes control software. Figs. 4 and 5 provide a conceptual software architecture for managing spring-loaded enclosures according to the present invention.

As shown in Fig. 4, the system includes cursor position logic 100 which maintains information concerning the current position of the cursor within the desktop area on the display. Also, drag logic 101 monitors the mouse actions including the mouse button and movement of the mouse to indicate the execution of a drag operation. Obviously with pointing devices other than a mouse, a drag operation may be indicated by a variety of user actions.

The system also includes timer logic 102 for detecting a pause of the cursor over a particular identifier in the spring-loaded enclosure management routine. Also, the timer 102 may be used for other purposes.

Also included in the control software is spring-loaded enclosure management logic 103. This management logic maintains a list of temporary windows referred to as the "Sprung Stack", and an indicator of the Top temporary window in the Sprung Stack. The Sprung Stack consists of a set of pointers to records that identify the state of the temporary windows. These records are referred to as Sprung Records.

The system further includes window management logic 104 which performs a variety of functions known to those of skill in the art for managing windows. This window management logic includes a system for opening and closing windows on the desktop in response to pointer actions, and maintains a desktop Window List. The desktop Window List comprises a list of windows that are opened on the desktop, their positions on the desktop, and other characteristics of the window, including the location, the types of windows, and information about icons, if any, enclosed by the window. Further, enclosure windows in the list point to a window record that contains information about the enclosures that are represented by identifiers in the windows, and such other information as is needed.

In cooperation with the drag logic 101, the window management logic 104 maintains a parameter referred to as the Current Window, which indicates the window within which the cursor currently resides. Also, the window management logic 104 maintains a parameter referred to as the Last Window which indicates the last window which the cursor was over, for instance if the cursor moves from inside a window to outside a window. Finally, the window management logic maintains a record referred to as the Current Enclosure which indicates the identifier that the cursor is presently positioned over.

A final component of the control software is known as the drawer management logic 105. The drawer management logic manages windows which are maintained on the screen with a desk drawer metaphor. In particular, the windows are positioned along the periphery of the desktop. They can be slid off the desktop leaving only a drawer identifier on the perimeter of the desktop display. When the desk drawer window is opened in response to a cursor action executed by the user, the window slides onto the screen obscuring any windows underneath it. When the user is done with the window, it can be slid back offscreen to reveal the underlying windows. The desk drawer management logic 105 maintains a Threshold parameter which indicates the distance from the perimeter of the desktop within which a cursor movement will result in opening of the drawer, and a Current Drawer parameter indicating the drawer within which the cursor is currently positioned.

The drawer management logic is described in detail in our co-pending U.S. patent application entitled COMPUTER SYSTEM WITH GRAPHICAL USER INTERFACE INCLUDING DRAWER-LIKE WINDOWS, invented by Thomas J. Conrad and Elizabeth Moller.

Fig. 5 illustrates the basic data records maintained by the spring-loaded enclosure management logic 103 and the window management logic 104. In particular, the window management logic 104 maintains a desktop window list 110. It includes an entry for the desktop with a set of pointers to a list of identifiers within the desktop. Also, each enclosure which is opened as a window within the desktop is added to the list as indicated. Thus, the list includes window 1, window 2, window 3, etc. Associated with each window, e.g., window 1, is an icon pointer which points to a record of

identifiers, e.g., record 111. Each identifier within window 1 is maintained in a list 111. Thus, identifier 1 includes a pointer to an enclosure record 112, which indicates the type of enclosure, whether the enclosure has been opened on the desktop, the location on the desktop of the identifier, the location in memory of the object represented by the identifier, etc. If the identifier has been opened on the desktop, then information about its location will be included in the window list 110.

The spring-loaded enclosure management logic 103 maintains a Sprung Stack 113. This includes a list of temporary windows, window X, window Y, and so on, which have been opened during a drag operation according to the present invention. Each entry in the Sprung Stack points to a Sprung Record 114. The Sprung Record maintains such information about the temporary window as whether the window was previously opened on the desktop and, if so, where; the location on the desktop of the temporary window; a pointer to a list of identifiers (e.g., 111) for the temporary window; etc.

A functional flow chart for executing the spring-loaded enclosure management is provided with reference to Figs. 6-13, in which Fig. 6 is the Main Loop. The Main Loop shown in Fig. 6 begins at block 600 where it monitors the mouse button. If the mouse button is not down, the algorithm loops at block 600. If the mouse button is down, then the algorithm passes through the Drawer Stuff routine which handles clicks of the mouse button in drawers represented by block 601 which is shown in detail in Fig. 12. After the Drawer Stuff routine 601, the algorithm tests to determine whether the cursor is over an object (block 602). If it is not over an object, then the algorithm handles other clicks and

cursor operations (block 603) and loops to block 600. If the cursor is over an object at block 602, then the algorithm monitors the mouse button (block 604). If the mouse button does not remain down, then a Select Object routine is executed, as indicated at block 605, and the algorithm loops to block 600. The select object routine 605 may result in a variety of operations as known in the art, such as opening an application window.

If the mouse button remains down at block 604, then a drag operation is indicated. In this case, the routine creates a grey outline (referred to as an "altered view" above) of the object and attaches the grey outline to the cursor. Also, the Last Window parameter is set equal to the Current Window parameter (block 606).

After block 606, the mouse button is monitored (block 607). If the mouse button is released, then the algorithm branches to the Finish Drag routine represented by block 608 which is shown in detail in Fig. 13. If the mouse button remains down, then the algorithm proceeds through the Drag Over Window routine represented by block 609 shown in Fig. 7. After the Drag Over Window routine at block 609, the algorithm loops to block 607 to monitor the drag operation.

Fig. 7 illustrates the Drag Over Window routine represented by block 609 in Fig. 6. The Drag Over Window routine is started from block 607. First, it tests whether the Current Window parameter is equal to the Last Window parameter (block 700). If it is not equal, the border of the last window has been crossed and the algorithm branches to the Window End Drag routine represented by block 701 which is shown in detail in Fig. 10. If the Current Window remains equal to the Last Window (the cursor remains within the boundary of the window), then the algorithm branches to the In Window

routine represented by block 702, as shown in detail in Fig. 8. From the Window End Drag routine of block 701, and the In Window routine of block 702, the algorithm proceeds through block 703 where Last Window is again set to Current Window. Next, the algorithm tests whether the Current Window is a drawer (block 704). If it is not a drawer, then the routine returns to block 607 of Fig. 6. If the Current Window is a drawer, then the algorithm tests whether the Current Drawer is equal to the Current Window (block 705). If not, then the cursor has moved out of the Current Drawer, and the Current Drawer is closed (block 706). The Current Drawer parameter is set to the Current Window in block 707 and the algorithm loops back to block 607 of Fig. 6. If, in block 705, the Current Window is not a drawer, the algorithm returns to block 607 of Fig. 6.

Fig. 8 illustrates the In Window routine represented by block 702 of Fig. 7. The In Window routine is entered from block 700 of Fig. 7. First, the algorithm tests whether the Current Window is equal to Current Drawer (block 800). If it is the Current Drawer, then the In Drawer routine represented by block 801 and shown in detail in Fig. 11 is executed. If the Current Window is not equal to the Current Drawer in block 800, and from the output of the In Drawer routine in block 801, the algorithm branches to block 802, where the algorithm determines whether the cursor is over an enclosure icon. If it is not over an enclosure icon, then the algorithm returns to block 703 of Fig. 7. Otherwise, the algorithm branches to block 803, where a timer is set to zero, and the Last Enclosure parameter is set equal to the Current Enclosure. Next, the algorithm monitors whether the cursor remains over the Current Enclosure by testing whether the Last Enclosure remains equal to the Current

Enclosure in block 804. If it does not remain over the Current Enclosure, then the timer is tested (block 805). If the timer is equal to zero, then the algorithm branches to block 703 of Fig. 7. If it is not equal to zero, the timer is first reset to zero in block 806 and then returns to block 703 of Fig. 7.

If, in block 804, the cursor remains over the Current Enclosure, the algorithm tests whether the timer is equal to zero in bock 807. If the timer is not equal to zero, the algorithm tests whether the timer has expired in block 808. If it has expired, then the Spring Open Window routine represented by block 809 and shown in detail in Fig. 9 is executed. After the Spring Open Window routine in block 809, the algorithm returns to block 703 of Fig. 7.

If in block 807 the timer was equal to zero, then the timer is started (block 810) and the algorithm loops to block 804 to begin monitoring whether the cursor remains over the enclosure until expiration of the timer.

Fig. 9 illustrates the Spring Open Window routine corresponding to block 809 of Fig. 8. The algorithm is entered from block 808 of Fig. 8. The first step is to create the Sprung Record for the particular window being sprung open (block 900). Next, the algorithm determines whether the window being sprung open is already open on the desktop (block 901). If it is already open, then the existing window size and position are saved in the Sprung Record (block 902). Next, the existing window is removed from the screen (block 903). Coupled with removing the existing window, a zoom operation may be executed to graphically illustrate movement of the window being closed to the new position.

If the window to be sprung open was not already open at block 901, or after block 903, the algorithm opens a

temporary window centered about the cursor position (block 904). Next, the temporary window centered about the cursor position is tested to determine whether it is partially offscreen (block 905). If it is partially offscreen, then the window is moved onto the screen (block 906). After block 906, or if the window is completely on the screen at block 905, then the algorithm tests whether the window is too big for the sprung open routine (block 907). If the window is too big then it is resized to fit on the monitor that the cursor is currently on, leaving several pixels of free space around the window (block 908).

After block 907, if the window is not too big, or after it is resized in block 908, the algorithm loops to block 909 where the window reference (description of the temporary window) is stored in the Sprung Record. Next, the Sprung Record is pushed onto the Sprung Stack (block 910). After block 910, the algorithm returns to block 703 of Fig. 7.

Fig. 10 illustrates the Window End Drag routine represented by block 1001 of Fig. 7. The Window End Drag routine is entered from block 700 of Fig. 7. The first step is to determine whether the Last Window is a slid open drawer in block 1001. If it is a slid open drawer, then the drawer is closed (block 1002).

If the Last Window is not an open drawer, or after the drawer is shut in block 1002, then the algorithm tests whether the Last Window is the Top Window in the Sprung Stack (block 1003). If it is not, then the algorithm returns to block 1003 of Fig. 7 or to block 1300 of Fig. 13.

If at block 1003 the Last Window is the Top window in the sprung stack or if the algorithm is entered from

block 1301 of Fig. 13, then the Last Window is popped off the Top of the Sprung Stack (block 1004).

After the Sprung Record is popped off of the Top of the Sprung Stack, the temporary window corresponding to the Top record is closed (block 1005). Next, the algorithm determines whether the Top window (being popped off the stack) needs to be re-opened (block 1006). This occurs if the Top was open at another location on the desktop before the temporary window was opened during the drag operation. If so, the Top is re-opened at the location indicated in the Sprung Record, and then the algorithm determines whether it should be resized (block 1007). If it needs to be resized, then the resize operation is executed (block 1008). If the Top that is being popped off the top of the Sprung Stack does not need to be resized, or after the resizing in block 1008, then the new Top of the Sprung Stack is opened (block 1009). After block 1009, or if the window does not need to be re-opened from block 1006, the algorithm returns to block 1003 of Fig. 7 or to block 1300 of Fig. 13.

Fig. 11 illustrates the In Drawer routine corresponding to block 801 of Fig. 8. The In Drawer routine in entered from block 800 of Fig. 8 and first tests whether the cursor is within the Threshold of the drawer management logic (block 1100). If it is not, then the algorithm returns to block 802 of Fig. 8. If it is within the Threshold, then the drawer is slid open one notch (block 1101). After block 1101, the algorithm returns to block 802 of Fig. 8. By keeping the cursor in this position, the user causes the drawer to gradually slide open.

Fig. 12 illustrates the Drawer Stuff routine entered from block 600 of Fig. 6. It first tests whether the button remains down (block 1200). If it is down, then

the algorithm returns to block 602 of Fig. 6. If the button has been released, then the algorithm tests whether the Current Drawer is equal to the Current Window (block 1201). If the cursor remains within the Current Window, the algorithm determines whether the click (release detected in block 1200) occurred in the title bar of the opened drawer (block 1202). If not, the algorithm returns to block 602 of Fig. 6. If the click was in the title bar, or if the click was not in the Current Drawer as indicated at block 1201, then the Current Drawer is removed from the screen (block 1203). Next, the algorithm determines whether the Current Window is a drawer at all (block 1204). If not, it returns to block 602. If the Current Window is a drawer, then the drawer is opened (block 1205). This occurs when a drawer is open and another drawer is clicked.

Fig. 13 illustrates the Finish Drag routine corresponding to block 608 of Fig. 6. The Finish Drag routine is entered from block 607 of Fig. 6. The algorithm first determines whether the Sprung Stack is empty in block 1300. If it is not empty, then the Window End Drag routine is executed as indicated at block 1301. The Window End Drag routine of block 1301 enters the routine of Fig. 10 at block 1004. After the Window End Drag routine of block 1301, the algorithm returns to block 1300. If the Sprung Stack was empty at block 1300, then the algorithm returns to block 600 of Fig. 6.

III. Alternate Spring Open Gestures (Figs. 14A-B and 15A-B)

Figs. 14A-B and 15A-B illustrate alternative mouse gestures that may be used for springing open enclosures according to the present invention. The preferred system, as illustrated above, springs open enclosures in

response to a pause of the cursor over the enclosure to be opened, as described with reference to Fig. 8. Alternative systems may be implemented that spring open enclosures based on other pointer gestures. Two examples are shown in Figs. 14A-B and Figs. 15A-B.

In Figs. 14A-B, enclosure icon 1400 and a dragged icon 1401 are shown. The enclosure icon 1400 has a hot region, or temporary window open region, 1402 and a main box 1404. If the cursor is moved into the hot region 1402, as illustrated in Fig. 14B, then the sprung open enclosure will be open. If the cursor does not hit the hot region 1402, then no action occurs. If the mouse button is released over the main box 1404, then the dragged object goes inside the enclosure.

In Figs. 15A and 15B, another alternative sequence is shown. In this sequence, a dragged icon 1501 is dragged over a folder. When this occurs, a select icon appears, such as an opened folder icon 1502 with a split pie symbol. The split pie has a first side 1503 and a second side 1504. If the user moves the cursor downward to the second side 1504, as illustrated in Fig. 15B, then the sprung open enclosure is opened. Alternatively, if the user moves the cursor upward into the first side 1503, then some other action may occur. If user moves the cursor through the split pie, then the select icon is removed and the original icon reappears. As before, if the mouse button is released over the opened folder, then the dragged object goes inside the folder.

Those skilled in the art will appreciate that there are a variety of techniques for indicating the desire to spring open an enclosure during a drag operation.

IV. Conclusion

A new behavior of the graphical user interface has been provided which allows a user to open and close enclosures, such as folders in the Macintosh Finder™ environment, while dragging some other object. When the user pauses, gestures, or rolls over a hot spot on the object during the drag, a temporary window corresponding to that object is opened on top of the cursor. This allows the user to browse inside the enclosure and possibly open a hierarchy of enclosures contained within the newly opened window during the drag operation. The user thus has access to the entire storage system hierarchy during a drag operation. By using the spring loaded enclosure mechanism, the user is left free to browse while dragging, rather than being forced to set up source and destination windows before a drag begins. This greatly improves the basic copy and move functions provided by the graphical user interfaces based on windows and icons.

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.